View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by ZENODO

An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order

Hannes Gross, Stefan Mangard, Thomas Korak

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria
{firstname.lastname}@iaik.tugraz.at

Abstract. Passive physical attacks, like power analysis, pose a serious threat to the security of digital circuits. In this work, we introduce an efficient side-channel protected Advanced Encryption Standard (AES) hardware design that is completely scalable in terms of protection order. Therefore, we revisit the private circuits scheme of Ishai *et al.* [13] which is known to be vulnerable to glitches. We demonstrate how to achieve resistance against multivariate higher-order attacks in the presence of glitches for the same randomness cost as the private circuits scheme. Although our AES design is scalable, it is smaller, faster, and less randomness demanding than other side-channel protected AES implementations. Our first-order secure AES design, for example, requires only 18 bits of randomness per S-box operation and 6 kGE of chip area. We demonstrate the flexibility of our AES implementation by synthesizing it up to the 15^{th} protection order.

Keywords: domain-oriented masking, private circuits, threshold implementations, ISW, side-channel analysis, DPA, hardware security, AES

1 Introduction

The increasing number of interconnected devices demand security not only on a cryptographic level but also on a physical level. Without countermeasures against physical attacks, devices are defenseless against attackers which have physical access. An attacker can easily extract device internal secrets by measuring the power consumption [14] or the electromagnetic emanation [19] of the device during security critical operations.

The most promising approach to achieve resistance against passive physical attacks is to make sensitive computations independent from the processed data by using so-called masking schemes. There exist many masking schemes, the scheme of Goubin *et al.* [10], or Ishai et al.'s private circuits [13], and the Trichina gate [22]. However, the aforementioned schemes have been shown to be vulnerable against glitches and thus rigorous care has to be taken during the implementation to avoid leakage caused by glitches.

There exist masking schemes that are inherently immune against glitches. The most popular scheme is the threshold implementation (TI) masking scheme introduced by Nikova *et al.* [18]. It has been extensively researched and extended by Bilgin *et al.* [1, 4] during the last years. There exist many protected hardware implementations that are based on TI [2, 3, 17].

Recently, Reparaz *et al.* introduced the Consolidated Masking Scheme [20] (CMS). One interesting aspect of the CMS scheme is the possibility to reduce the number of required input shares of TI from $td + 1$ to $d + 1$, where d corresponds to the attack order and t is the algebraic degree of the function that should be protected. At CHES 2016, De Cnudde *et al.* [7] demonstrated the suitability of using only $d+1$ shares on an AES hardware design. The design requires less chip area than related work, but at the cost of an increased randomness demand compared to $td + 1$ TI. More specifically, the CMS scheme requires $(d + 1)^2$ random bits for protecting one $GF(2^n)$ multiplication as required multiple times for the AES S-box.

Producing a high amount of random numbers in hardware, however, is not trivial and goes hand in hand with an increased chip area usage, a higher energy consumption, and has also a negative influence on the throughput of a design. Therefore, for the efficiency of masked implementations the randomness demand is crucial.

Our Contribution. In this work [1], we demonstrate how the randomness requirements for $d+1$ masking can be lowered from $(d+1)^2$ to only $d(d+1)/2$. In order to achieve this, we revisit the private circuits scheme [13] which is known to be vulnerable to glitches. We perform a similar approach under the premise of glitches, and demonstrate how to achieve d^{th}-order protection in the presence of glitches for the same randomness cost and without losing genericity. We show the suitability of our approach by implementing a d^{th}-order protected AES-128 encryption-only hardware design. Our first-order AES implementation requires only 18 fresh random bits per S-box calculation, which is a third of the random bits of the CMS implementation of De Cnudde *et al.* [7]. Our AES design is also very compact in terms of chip area and requires only 6 kGE of chip area and 246 clock cycles per encryption. Furthermore, our approach is generic in terms of protection order, allowing our AES design to be synthesized for any desired protection order. The number of required clock cycles per encryption, however, is independent of the protection order. We demonstrate the genericity of our design by stating post-synthesis hardware results up to the 15^{th} protection order. The VHDL source code of the generic AES design is published online [11], which we hope will help future research and make comparisons easier.

2 Private Circuits and the ISW Transformation

The original idea of Ishai *et al.* [13] was to build a so-called private circuit compiler that can transform arbitrary circuits into circuits that resist passive physical attacks, like chip probing and side-channel analysis, up to a protection order d. For this purpose, the circuit's data signals are first split into a number of shares, which when recombined through addition over $GF(2)$ result in the original value. The sharing is done based on uniformly distributed random numbers. A sharing of a signal x can be written as shown in Equation 1, where the shares are denoted by capital letters with the name of the shared signal in the subscript index.

[1] An earlier version of this work has been published online [12] under the title "Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order".

$$x = \underbrace{A_x + B_x + C_x + \ldots}_{d+1 \text{ shares}} \tag{1}$$

The security of masking schemes is typically shown in the so-called d-probing model. A masking scheme provides security of order d in this model, if each combination of up to d signals is independent of all unshared intermediate signals. It was demonstrated by Faust *et al.* [8] and Rivain *et al.* [21] that there indeed exists a relation between the number of probed wires in the d-probing model and the attack order for a differential power analysis (DPA) attack.

The security of the sharing of x in Equation 1 against a d-probing attacker follows from the fact that the attacker only gets access to the $d+1$ shares of x (A_x, B_x, ...) but not to x itself. The circuit is secure against d probes, as long as no signal in the circuit contains a combination of more than one share of x. To keep this share independence, also all gates of the circuit are required to fulfill this requirement.

The basic idea of the ISW transformation in order to achieve this, is therefore to transform the original circuit in a way that it only consists of protected NOT and AND gates. While the protected implementation of the NOT gate is straightforward and only requires the negation of one share (see Equation 2), the protected implementation of the AND gate is more difficult and requires the introduction of fresh randomness to fulfill the independence requirement.

$$\neg x = \neg A_x + B_x + C_x + \ldots \tag{2}$$

AND Gate. For the correct and secure realization of the AND gate in the ISW scheme (with x and y as the input and q as the output), Equation 3 needs to be expanded, securely evaluated, and compressed again to $d+1$ output shares.

$$q = xy = (A_x + B_x + C_x + \ldots)(A_y + B_y + C_y + \ldots) \tag{3}$$

To achieve independence during the compression, some terms need to be first remasked by using fresh randomness denoted by Z shares in the following. Equation 4 shows an example for an ISW implementation of an AND gate for $d = 2$ in our notation. For a general description of the compression algorithm see [13], for details. The correctness of the AND gate in Equation 4 is given because all random Z shares appear exactly twice in additive manner, and the rest of the terms are the one of the expanded Equation 3.

$$\begin{aligned} A_q &= A_x A_y + Z_0 + Z_1 \\ B_q &= B_x B_y + (Z_0 + A_x B_y + B_x A_y) + Z_2 \\ C_q &= C_x C_y + (Z_1 + A_x C_y + C_x A_y) + (Z_2 + B_x C_y + C_x B_y) \end{aligned} \tag{4}$$

For the security of Equation 4, the order in which the terms are summed up is critical. While the calculation order can be easily controlled for software implementations, the order in which the terms are summed up cannot so easily be controlled in the combinatorial logic of hardware implementations.

Like many other masking schemes, the private circuits approach is therefore considered to be vulnerable to so-called glitches [15]. Glitches are caused in the combinatorial path of hardware circuits because the electric signals do not propagate with unlimited speed. Instead signal arrival times and delays at the logic gates can cause several changes at the output of a gate before the gate output reaches its final state (for more details see, *e.g.* , [16]). Digital designers also have only marginal influence on the exact placement of the logic gates, the signal timings, and the order in which the signals are combined. A secure masking scheme thus needs to be inherently immune against glitches without relying on correct placement of the gates and signal timings.

Since there exist secure ISW implementations in software, a straightforward approach of its implementation in hardware would be to emulate the behavior of a processor running the ISW transformed software. As a result, the output of each AND and each XOR operation would be first stored in a register before any further processing is performed. However, this approach is neither very resource friendly nor efficient in terms of throughput.

In the next section, we thus introduce a secure construction of a masked AND gate in hardware and argue its security in the d-probing model for the case that glitches are taken into account. Our masked AND gate uses the same multiplication terms as the ISW AND gate, and has the same randomness requirements and a generic structure. However, in contrast to ISW, the introduced masked AND gate is resistant to glitches and has a balanced gate distribution which is desirable in order to minimize the delay of a hardware implementation. We start our construction and security argumentation for a first-order secure masked AND gate before we generalize the concept to arbitrary protection orders.

3 A Glitch-Resistant Masked AND Gate

The basic idea behind our glitch-resistant masked AND gate, is to split the calculation of Equation 3 into independent share *domains*. Each share of a signal is associated with one specific domain. This is also reflected in the notation that is used in this paper. The shares A_x and B_x of a data signal x, for example, are associated with the domains labeled A and B, respectively.

The AND gate uses $d + 1$ shares per signal in order to achieve d^{th}-order security and there are $d + 1$ domains in this case. The intuition behind this approach is to keep the shares of all domains independent from shares of other domains. This independence ensures d^{th}-order security according to the d-probing model when considering glitches.

The critical parts of the circuit, are the parts that need to process inputs from multiple domains. In this case dedicated measures need to be taken before the terms can be securely served as inputs of a domain. By adding a fresh random share Z to these terms, the terms can be reassociated to a targeted domain. Furthermore, the usage of a register in this case prevents that glitches propagate from one domain to the another domain.

We first start with the introduction of the glitch-resistant AND gate for first-order security before this approach is extended to arbitrary protection orders.

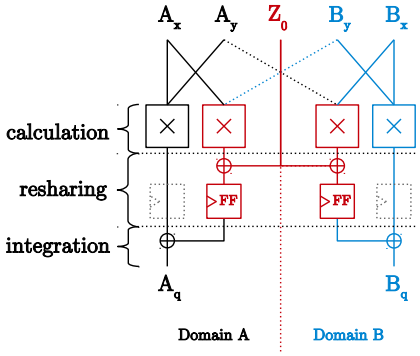

Fig. 1. First-order secure AND gate

3.1 1st-Order Secure AND Gate

A first-order secure AND gate (see Figure 1) consists of two domains labeled A and B. The inputs x and y are provided to the AND gate by the shares A_x and B_x, and A_y and B_y, respectively. The sharings for x and y are required to be uniformly random and independent of each other. The AND gate returns the shares A_q and B_q of the output q. The calculations are performed in three steps in order to map the input shares to the output shares. We refer to these steps as *calculation*, *resharing* and *integration*.

Calculation: In the first step, the actual calculation of the logic function (expanded Equation 3) is performed and the terms A_xA_y, A_xB_y, B_xA_y and B_xB_y are calculated. The terms that can be directly associated with one domain (inner-domain terms) are the terms A_xA_y and B_xB_y, respectively. These terms are not critical from a security point of view. Any computation on inner-domain terms associated with one specific domain, only lead to outputs that again depend only on shares associated with this domain.

In case of terms that contain different domain labels (cross-domain terms), there is less freedom. In fact these calculations are only secure for independently shared input signals. If shares of the same signal would be combined for example, the independence would be trivially broken. For example, the term A_xB_x would leak information about x. However, shares associated with different domains that correspond to different signals of the unprotected circuit can be combined without violating the requirement for d^{th}-order security. In fact, there is no leakage about x or y when calculating A_xB_y. This results from the requirement that x and y are independently shared. There is also no leakage caused by B_xA_y for an independent sharing of x and y. Cross-domain terms of the AND gate that can not directly be associated with one domain are plotted red in Figure 1.

Resharing: The integration of the cross-domain terms into a specific domain is prepared in the resharing step. By adding a fresh random Z share to these terms, the term becomes statistically independent from all other shares and can therefore be associated with any arbitrary domain in the next step. In case of the 1st-order secure AND gate, the same fresh share Z_0 is used for the resharing of the product terms A_xB_y and B_xA_y.

This does not lead to any first-order leakage, because a probing attacker restricted to one probing needle cannot find a single signal in the AND gate that correlates to the unshared inputs x and y or the output q.

In order to prevent that any glitch propagates through the resharing step, a register is included as last part of the resharing step. The two registers in grey dotted lines are optional registers and are only required for pipelining purposes but not for the security of the AND gate.

Integration: During the integration phase, the reshared cross-domain terms are added to the inner-domain terms, which concludes the calculation of the AND gate. Please note that this addition leads to glitches at the XOR gate at the output of the domain. However, as the resharing step finishes with a register no glitches can occur that depend on x or y. In terms of correctness, it is important to point out that the fresh share Z_0 becomes part of both domains. Hence, it holds that $q = A_q + B_q$.

In summary, the security against a first-order probing attacker is given because each domain contains either inner-domain terms that contain only shares that are already associated with one specific domain, or cross-domain terms that are reshared with a fresh random Z share which is only used once in each domain. An attacker thus always needs to combine at least two signals to get one signal that depends on one of the independently shared inputs x or y.

3.2 Higher-Order Secure AND Gate

The first-order AND gate can be extended to arbitrary protection orders. The generalization requires to first extend the *calculation* step to produce a correct sharing with $d + 1$ shares for any given protection order d. In the *resharing* phase it needs to be ensured that the fresh random Z shares are distributed over the domains in a way that (1) each cross-domain term is reshared with a Z share that is unique inside the targeted domain, and (2) none of the signal combinations created in the *integration* phase reveals more than the inner-domain terms or shares of the respective domain.

Calculation: The same rules as for the first-order AND gate apply for the higher-order generalization. Again, any combination of shares can be safely used inside their associated domain without any restrictions. Cross-domain terms, however, require independently shared signals to prevent the case that two shares of the same sharing are combined. This ensures that by probing a cross-domain term, the attacker does not learn more about the inputs x and y then when probing a share of x and y directly.

The calculation step can be generalized for $d + 1$ input shares as shown in Equation 5. Each row of this formula stands for one domain with a dedicated label calculating one share of the output q. The terms in the diagonal (bold) are the inner-domain terms containing only shares from one specific domain and hence only leak about shares of this domain. The cross-domain terms do not leak more information on the inputs x and y then when probing one share of x and one share of y directly. Hence, with this formula the sharing for the *calculation* step for the AND gate resists a d-probing attacker for an arbitrary numbers of shares. An example for a second-order AND gate is given in Figure 2.

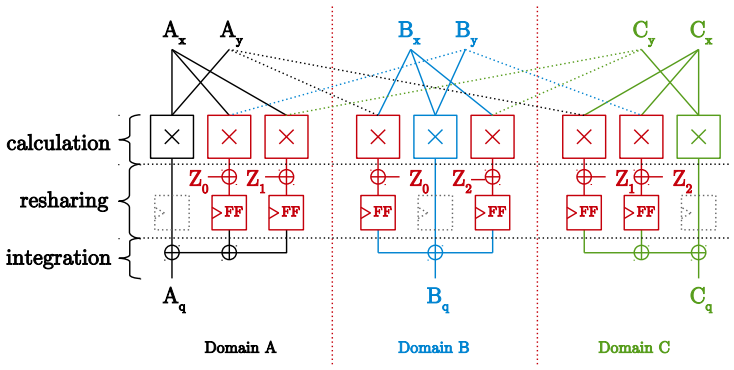

Fig. 2. Second-order secure AND gate

$$
\begin{aligned}
\underbrace{\mathrm{A_q}}_{\mathrm{Q_0}} &= \underbrace{\boldsymbol{A_x A_y}}_{t_{0,0}} + \underbrace{(A_x B_y + Z_0)}_{t_{0,1}} + \underbrace{(A_x C_y + Z_1)}_{t_{0,2}} + \underbrace{(A_x D_y + Z_3)}_{t_{0,3}} + \underbrace{(A_x E_y + Z_6)}_{t_{0,4}} + \dots \\
\underbrace{\mathrm{B_q}}_{\mathrm{Q_1}} &= \underbrace{(B_x A_y + Z_0)}_{t_{1,0}} + \underbrace{\boldsymbol{B_x B_y}}_{t_{1,1}} + \underbrace{(B_x C_y + Z_2)}_{t_{1,2}} + \underbrace{(B_x D_y + Z_4)}_{t_{1,3}} + \underbrace{(B_x E_y + Z_7)}_{t_{1,4}} + \dots \\
\underbrace{\mathrm{C_q}}_{\mathrm{Q_2}} &= \underbrace{(C_x A_y + Z_1)}_{t_{2,0}} + \underbrace{(C_x B_y + Z_2)}_{t_{2,1}} + \underbrace{\boldsymbol{C_x C_y}}_{t_{2,2}} + \underbrace{(C_x D_y + Z_5)}_{t_{2,3}} + \underbrace{(C_x E_y + Z_8)}_{t_{2,4}} + \dots \\
\underbrace{\mathrm{D_q}}_{\mathrm{Q_3}} &= \underbrace{(D_x A_y + Z_3)}_{t_{3,0}} + \underbrace{(D_x B_y + Z_4)}_{t_{3,1}} + \underbrace{(D_x C_y + Z_5)}_{t_{3,2}} + \underbrace{\boldsymbol{D_x D_y}}_{t_{3,3}} + \underbrace{(D_x E_y + Z_9)}_{t_{3,4}} + \dots \\
\underbrace{\mathrm{E_q}}_{\mathrm{Q_4}} &= \underbrace{(E_x A_y + Z_6)}_{t_{4,0}} + \underbrace{(E_x B_y + Z_7)}_{t_{4,1}} + \underbrace{(E_x C_y + Z_8)}_{t_{4,2}} + \underbrace{(E_x D_y + Z_9)}_{t_{4,3}} + \underbrace{\boldsymbol{E_x E_y}}_{t_{4,4}} + \dots \\
&\vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \;\ddots
\end{aligned}
\tag{5}
$$

Resharing: A core property for the generalization of this AND gate implementation is how the required fresh random Z shares can be efficiently distributed among the cross-domain terms in a correct manner. From Equation 5 it can be seen that there are exactly $d(d+1)$ cross-domain terms which need to be reshared. It is also important to note that there are exactly two cross-domain terms that combine shares from the same two domains. For example shares from domain A and B are only combined in the terms $A_x B_y$ and $B_x A_y$. We use the same fresh Z share for cross-domain terms that combine shares from the same two domains. Hence, we use $d(d+1)/2$ fresh shares for a d$^{\text{th}}$-order AND gate, which is the same amount as in the ISW scheme.

Since no probing of any intermediate signal created in the *calculation* phase contains more than one share of each input x or y, and in the *resharing* phase we add fresh random shares to the cross-domain terms, no advantage to a d-probing attacker is given during these phases.

Integration: In the integration phase, the terms associated with each domain are added up at the output of the AND gate. Because a digital designer has no influence on the sequence in which these terms are added up (without forcing it through registers), the higher-order secure AND gate needs to provide probing security for each possible partial sum of these terms. In particular, it has to be taken care of that each of these possible partial sums an attacker could probe reveals only the shares of the domains she is probing in. This is ensured by the resharing shown in Equation 5, where each Z share is only reused for cross-domain terms with the same domain association.

In order to exploit the reuse of Z shares, it would be necessary to probe in the two domains that use the cross-domain terms with the reused Z share. However, the two cross-domain terms that use the same Z share contain only the shares of the same domains. Hence, there is no advantage for the attacker due the reuse.

For example, the share Z_0 in Figure 2 is used on the terms A_xB_y and B_xA_y and these two terms only occur in the domains A and B. An attacker that probes any partial sum of the terms in A learns only about shares in domain A. When probing any partial sum of the terms in B, there is only information about shares associated with B. A second-order attacker that learns about partial sums in A and B learns about shares from the domains A and B in any case. The fact that the cross-domain terms A_xB_y and B_xA_y reuse Z_0 does not provide any advantage to an attacker.

Based on Equation 5, the fact that the AND gate fulfills d^{th}-order security can also be verified visually. In this matrix the diagonal terms are formed by the inner-domain terms. These inner-domain terms also divide the matrix into an upper and lower triangular matrix in which each of the fresh random Z shares is used exactly once. The triangle formed by the Z shares is mirrored along the diagonal. The mirroring of the Z shares ensures that each possible combination of partial sums from any two domains removes at most one fresh random share, and reveals only the shares associated with both domains. Because this applies for all combinations of partial sums of all different domains, an attacker restricted to d probes obtains at most d shares per signal. However, for this security argumentation to hold it needs to be always ensured that the sharings of the inputs x and y are independent.

The domain equations of the matrix in Equation 5 can also be written in closed form as shown in Equation 6.

$$\mathrm{Q_i} = t_{i,i} + \sum_{j>i}^{d} \left(t_{i,j} + Z_{(i+j*(j-1)/2)}\right) + \sum_{j<i}^{d} \left(t_{i,j} + Z_{(j+i*(i-1)/2)}\right) \tag{6}$$

This equation is also the basis for the scalable AES design in the next section. Furthermore, we note that the approach can be easily extended to arbitrary finite fields. Consequently, our glitch-resistant masked AND gate, which equals a multiplication in $GF(2)$, can be extended to arbitrary large $GF(2^n)$ multiplications by replacing the AND gates in the calculation step by GF multipliers. Operations that are linear over $GF(2^n)$ like XOR or logic negation, on the other hand, can be applied to the shares without domain crossings. We use this property for an efficient implementation of the AES S-box in the next section.

4 d^{th}-order Secure AES Implementation

To compare the efficiency of our approach with existing masked implementations, we implemented the AES-128 encryption-only design suggested by Moradi and Poschmann [17]. Moradi's design was also used and modified by Bilgin *et al.* [2, 3, 6] and recently by De Cnudde *et al.* [7] for a $d + 1$ share CMS TI.

The control path of our modified AES design consists of a linear-feedback shift register (LFSR), the round constant generation module (RCON), and some additional logic gates to generate the control signals (see [17] for more details). Our LFSR module has a cycle length of 23. In each round, the first 16 cycles are spent on *AddRoundKey* and *SubBytes*. Then there are four cycles used for *MixColumns* and to calculate the first four bytes of the next round key. Then there are two dummy rounds inserted to bring the state register in correct position for further processing before in the final cycle the ShiftRows transformation is performed. The datapath mainly consists of the S-box, the key and state registers which are implemented as shift registers, the *MixColumns* module, and some multiplexers.

4.1 AES S-box

The by far most complex and most security critical part of the AES implementation is the S-box. Figure 3 shows our design of a 1^{st}-order protected variant of Canright's [5] AES S-box design. The S-box consists of many linear operations like the linear mappings at the input and the output, the square scalers, the sub-field inverters, and the adders. These are the parts that can be implemented share-wise for both domains in a straightforward way. The Galois field multipliers with different field order form the non-linear parts of the S-box. Canright's S-box makes repeated use of a finite field isomorphism to express $GF(2^8)$ elements as multiple elements in lower subfields—down to eight elements in $GF(2)$. These $GF(2^n)$ multipliers are replaced by the generalization of the masked AND gate of Section 3 for GF multipliers. Therefore, the standard-cell library AND cells used for the calculation step in the masked AND gate are simply replace by the according GF multipliers.

To maximize the efficiency of the implementation, seven pipelining stages are added to the S-box. The pipelining registers are marked with circles and appear along the red and green dotted lines in Figure 3. Red dotted lines indicate multiplier related stages which are also labeled Stage 1-5 in order to refer to them more easily. The green marked registers are required to ensure independence in the presence of glitches for the inputs of the adjacent GF gates. To make the S-box secure and efficient at the same time, it is necessary to pinpoint all GF gates that have related input sharings. These gates need to be treated more carefully than the one with independent inputs. We now discuss the security of each multiplication stage individually which reveals that the additional pipeline stages (plotted in green) are required at multiplication stages 1, 2, and 3, but not at 4 and 5.

Stage 1. The $GF(2^4)$ gate in Stage 1 receives its inputs from the linear mapping at the S-box input. The linear mapping takes the 8-bit input shares A_x and B_x and linearly combines these eight bits inside their respective domain (see [5] for more details).

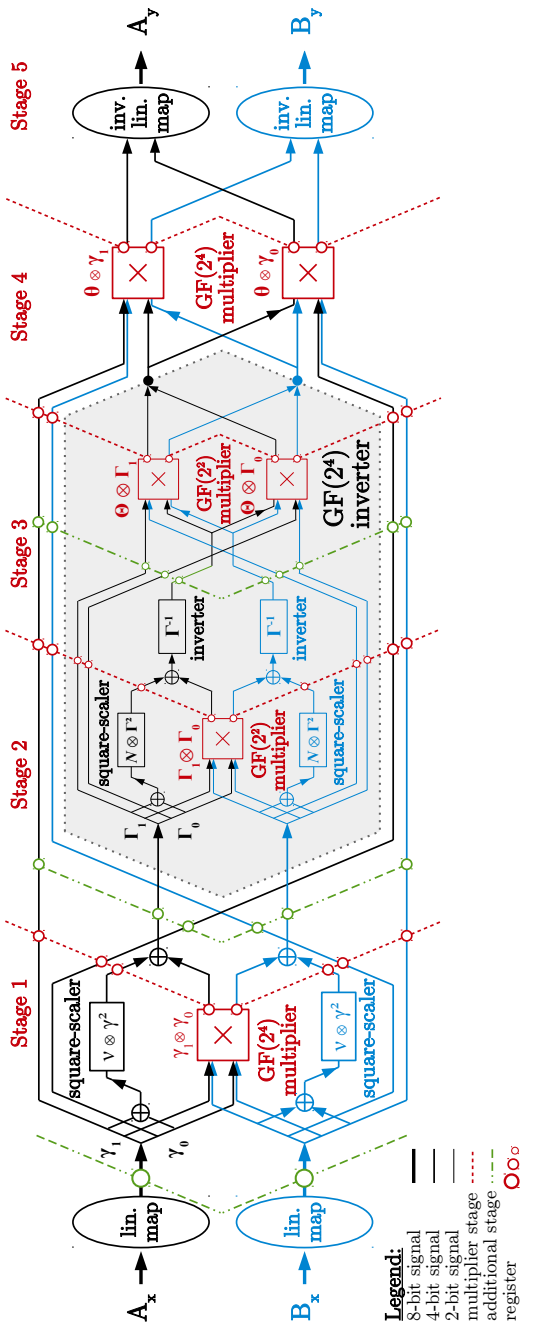

Fig. 3. First-order masked AES S-box with seven pipeline stages

Because of the different signal transition times and gate delays, it is therefore possible that the output of the linear mapping temporarily consists of bits with related sharing. Applying these bits directly to the GF gate from Figure 1—while the linear mapping has not yet settled—would thus violate the independence in the cross-domain terms associated GF multipliers. To avoid these glitches, registers are inserted after the linear maps to ensure the signals are settled before the bits are applied to the GF gate.

Stage 2 and 3. The situation is similar at Stage 2 and Stage 3. At these stages, glitches can occur from the combination of the square scaler outputs with the outputs of the GF gate. Again these glitches can be avoided by inserting pipelining stages at the marked positions in Figure 2.

Stage 4. For the GF gates in Stage 4, the inputs are the pipelined S-box inputs and the output of the GF gates of the previous stage. The output of the GF gate of Stage 3 originate from the inputs of the $GF(2^4)$ inverter which is remasked in Stage 1 (the masking is effective at latest at Stage 2). Therefore, the inputs of the Stage 4 GF gates are clearly independent and so no registers are required here.

Stage 5. The output mapping in this stage is again a linear transformation and uncritical as long as it is not followed by a nonlinear transformation that is unprepared for related sharing of its inputs. However, in our design of the AES core the output of the S-box is either stored in the key or state registers before it is used again, or fed into the S-box which is also uncritical because the input multiplier of either S-box variant is already prepared to process related input sharings.

The rest of the S-box is implemented according to the original Canright design but without some of its optimizations that would not be beneficial for our implementation. Canright's design, for example, reuses some temporary results in other parts of the S-box. Storing temporary results would lead to many additional pipelining registers for our design of the S-box and is therefore not suitable. For the generalization of the S-box to higher protection orders, the black (or blue) parts in Figure 3 are basically duplicated and the secure GF gates are generated as described in Section 3.

5 Implementation Results

All stated numbers are post-synthesis results for a 90 nm UMC Low-K process with 1.0 V power supply and 0.1 MHz clock frequency (in accordance with related work). Our designs are compiled with the Cadence Encounter RTL compiler version v08.10-s28_1 and routed with Cadence NanoRoute v08.10-s155. Please note that in general hardware result for different technologies, compiled and synthesized with different tool chains are difficult to compare. Furthermore, the functionality implemented by different modules is not always consistent with other implementations. The comparison of chip area results with related work should therefore be seen under this premise. To make comparison with our generic AES design easier for future work, we therefore decided on publishing the source code online [11].

Anyway, for a masked hardware design the number required fresh random bits is even more crucial for the efficiency of an implementation than the stated chip area of

Table 1. First-order secure AES-128 implementation results

Design/Module	Chip Area [%]	Chip Area [kGE]	Randomness [Bits/S-box]	Cycles	Throughput @0.1 MHz [Kbps.]
Our Implementation (90 nm)					
This work	100.0	**6.0**	**18**	**246**	**52**
S-box	37.3	2.2			
State registers	34.0	2.0			
Key registers	21.0	1.3			
Control, et cetera	7.7	0.5			
td+1 Threshold Implementations (180 nm)					
Moradi *et al.* [17]		11.0 / 10.8[a]	48	266	48
Bilgin *et al.* [2]		9.1 / 8.2[a]	44	246	52
Bilgin *et al.* [3]		8.1 / 7.3[a]	32	246	52
d+1 Threshold Implementations (45 nm)					
De Cnudde *et al.* [7]		6.7 / 6.3[a]	54	276	46

[a] This variant uses the *compile_ultra* flag which is not available in our tool chain.

the designs. The generation of fresh random bits with high entropy requires additional hardware and involves, e.g., complex analog circuitry or pseudo random number generators based on symmetric primitives. Both options have a critical influence on the chip area requirements, the energy budget, and on the delay or throughput.

First-order secure AES. Table 1 compares our first-order secure AES hardware implementation with existing related work. The $d + 1$ share designs of [7] with 6.7 kGE and our design with 6 kGE are smaller than the $td+1$ TI designs. The size difference mainly comes from the fact that $td + 1$ TI requires at least three shares for securely calculating non-linear functions while the first-order $d + 1$ share designs require only two shares.

In comparison with $d + 1$ TI design [7] which requires 54 random bits per S-box calculation, our design requires with 18 bits only a third of its random bits. Nevertheless, our design achieves the same throughput as the $td + 1$ TI design of Bilgin *et al.* with 52 Kbps for a 100 kHz clock and requires 14 bits less fresh randomness.

Second-order secure AES. In Table 2, a comparison of our second-order AES design with other second-order secure designs is given. In case of the $td + 1$ TI design the chip area was estimated by De Cnudde *et al.* [7]. Again, there is a noticeable gap between the $td + 1$ share design with about 14.9 kGE and the $d + 1$ share designs with about 10 kGE in terms of chip area resulting from the increased amount of shares (five shares versus three shares). Considering the randomness demand of the designs, our design requires 54 bits which is more than two times less than the $td + 1$ design with 126 fresh random bits, and three times less than the $d + 1$ TI design with 162 bits. In terms of throughput, our AES design requires 246 cycles instead of 276 cycles per encryption.

Table 2. Second-order secure AES-128 implementation results

Design/Module	Chip Area *[%]*	Chip Area *[kGE]*	Randomness *[Bits/S-box]*	Cycles	Throughput @0.1 MHz *[Kbps.]*
Our Implementation (90 nm)					
This work	100.0	**10.0**	**54**	**246**	**52**
S-box	45.1	4.5			
State registers	30.3	3.0			
Key registers	18.7	1.9			
Control, et cetera	5.9	0.6			
td+1 Threshold Implementation (estimated [7] , 45 nm)					
De Cnudde *et al.* [6]		*18.6 / 14.9*[a]	126	276	46
d+1 Threshold Implementation (45 nm)					
De Cnudde *et al.* [7]		10.5 / 10.3[a]	162	276	46

[a] This variant uses the *compile_ultra* flag which is not available in our tool chain.

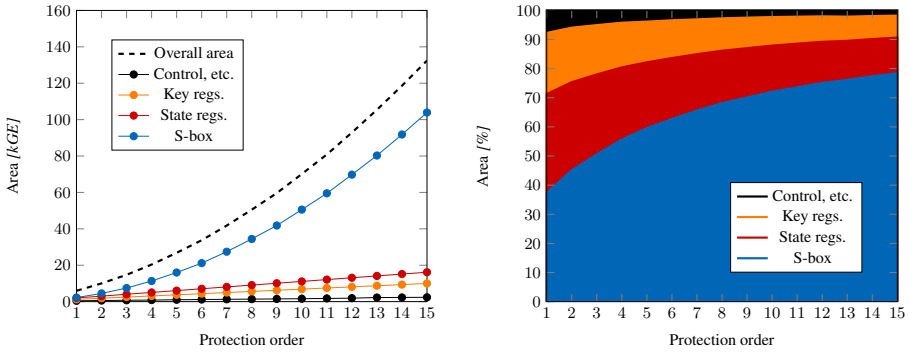

Fig. 4. Area requirements absolute (left) and in percent (right) per protection order

5.1 d^{th}-Order AES Implementation Results

The generic construction of our AES implementation not only allows the calculation of the number of required fresh random bits of $9d(d+1)$, but furthermore it is possible to synthesize the AES implementation for arbitrary protection orders by just changing one input parameter of our hardware design.

Figure 4 shows the post-synthesis area results for the different components in relation to the protection order. It can be observed that the state key and control logic requirements grow linearly with the protection order. The S-box and the contained GF gates grow quadratically. For the S-box, the size increases from 37.4% for the first-order implementation to about 78.5% for the 15$^{\text{th}}$-order. The relative size of the state and key register decrease from 34% and 21% to around 12.2% and 7.5%, respectively. The smallest amount of chip area is spent on the control logic which stays almost constant.

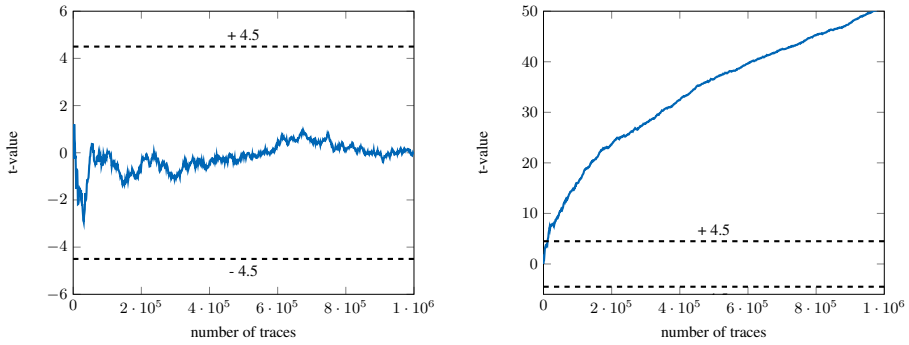

Fig. 5. First-order t-test (left) and second-order t-test (right) for first-order secure AES design

6 Side-Channel Evaluation

To show the resistance of our AES design against side-channel analysis attacks, different instances of the Welch's t-test are used (see Goodwill *et al.* [9] for details). The intention of this test is that for a side-channel secure implementation, a set of randomly picked (unshared) inputs should not show any statistically differences in the power traces for a set with constant inputs. For these two sets the so-called t value is calculated. If the t value is outside the confidence interval of ± 4.5 the null-hypothesis is rejected with confidence greater than 99.999% for large sizes of N.

Our evaluation approach is quite similar to what is checked in the d-probing model. Instead of using power trace values of, e.g., an FPGA implementation of our design, the t values of each individual signal are recorded for a post-synthesis netlist of our AES design during simulation. In comparison to an FPGA based validation this approach has three advantages: (1) the signals are completely noise free, (2) if any statistical differences are found, the violating signals can be directly pin-pointed, (3) if ASIC implementations are targeted, the synthesized netlist is closer to the final ASIC implementation than an FPGA implementation.

First-order AES design. The results of the first-order t-test for our first-order secure design are shown in Figure 5 (left) for up to one million traces. The t-value stays below the ± 4.5 border as required by the t-test to succeed. To demonstrate the soundness of our evaluation setup we also performed a second-order t-test. However, for the second-order t-test in a bivariate attack setting, performing individual t-tests for each signal separately is no longer feasible. The evaluation of each signal combined with every other signal for different points in time would take too long. Therefore, one single trace is calculated that sums up all signal transitions together. We then combine in each case two trace points over centered product pre-processing for all points in time within an eight clock cycles period (the delay of the S-box). As expected the t-tests fail with great confidence with t values clearly above the ± 4.5 border even for just a hundred traces.

Second-order AES design. The t-tests for the second-order AES design are illustrated in Figure 6. In both cases the t-tests do not indicate any leakage. We thus conclude that

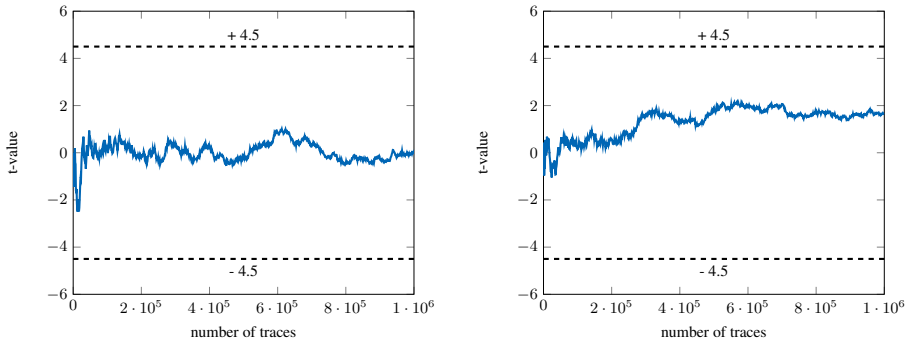

Fig. 6. First-order t-test (left) and second-order t-test (right) for second-order secure AES design

our implementation seems to be correct and secure in a bivariate second-order attack scenario.

7 Conclusions

In this work we introduced a generic hardware design of the AES. In contrast to existing implementations, our design is freely scalable in terms of resistance to side-channel analysis attacks. Because of its $d + 1$ share design principle it is also very efficient. With only 6 kGE of chip area, our design is the smallest published first-order (and beyond) masked AES implementation to this date.

Since the generation of random numbers with high entropy is a very demanding task for hardware implementations, we consider the randomness requirements to be even more decisive for the efficiency of a masked hardware implementation. In comparison with the recently published $d + 1$ share AES design [7], our design requires just $d(d + 1)/2$ fresh random shares instead of $(d + 1)^2$.

Acknowledgements.

This work has been supported by the Austrian Research Promotion Agency (FFG) under grant number 845589 (SCALAS). The HECTOR project has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 644052. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 681402).

References

1. B. Bilgin, J. Daemen, V. Nikov, S. Nikova, V. Rijmen, and G. Van Assche. Efficient and First-Order DPA Resistant Implementations of Keccak. In *CARDIS 2014*, LNCS. 2014.
2. B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. A More Efficient AES Threshold Implementation. In *AFRICACRYPT 2014*, volume 8469 of *LNCS*. 2014.
3. B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. Trade-Offs for Threshold Implementations Illustrated on AES. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 34(7), July 2015.
4. B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, and G. Stütz. Threshold Implementations of All 3x3 and 4x4 S-Boxes. In *CHES 2012*, volume 7428 of *LNCS*. 2012.
5. D. Canright. *CHES 2005*, chapter A Very Compact S-Box for AES. 2005.
6. T. D. Cnudde, B. Bilgin, O. Reparaz, V. Nikov, and S. Nikova. Higher-Order Threshold Implementation of the AES S-Box. In *CARDIS 2015*, 2015.
7. T. D. Cnudde, O. Reparaz, B. Bilgin, S. Nikova, V. Nikov, and V. Rijmen. Masking AES with d+1 Shares in Hardware. In *CHES 2016*, 2016.
8. S. Faust, T. Rabin, L. Reyzin, E. Tromer, and V. Vaikuntanathan. Protecting Circuits from Leakage: the Computationally-Bounded and Noisy Cases. In *EUROCRYPT 2010*, volume 6110 of *LNCS*. 2010.
9. G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi. A Testing Methodology for Side-Channel Resistance Validation. In *NIST Non-Invasive Attack Testing Workshop*, 2011.
10. L. Goubin and J. Patarin. DES and Differential Power Analysis The Duplication Method. In *Cryptographic Hardware and Embedded Systems*, volume 1717 of *LNCS*. 1999.
11. H. Gross. DOM Protected Hardware Implementation of AES. `https://github.com/hgrosz/aes-dom`, 2016.
12. H. Gross, S. Mangard, and T. Korak. Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. Cryptology ePrint Archive, Report 2016/486, 2016. `http://eprint.iacr.org/2016/486`.
13. Y. Ishai, A. Sahai, and D. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO 2003*, volume 2729 of *LNCS*. 2003.
14. P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. CRYPTO '99, London, UK, 1999. Springer-Verlag.
15. S. Mangard, T. Popp, and B. Gammel. Side-Channel Leakage of Masked CMOS Gates. In *CT-RSA 2005*, volume 3376 of *LNCS*. 2005.
16. S. Mangard and K. Schramm. Pinpointing the Side-Channel Leakage of Masked AES Hardware Implementations. In *CHES 2006*, volume 4249 of *LNCS*. 2006.
17. A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. EUROCRYPT'11. Springer-Verlag, 2011.
18. S. Nikova, C. Rechberger, and V. Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In *Information and Communications Security*, volume 4307 of *LNCS*. 2006.
19. J.-J. Quisquater and D. Samyde. ElectroMagnetic Analysis (EMA): Measures and Countermeasures for Smart Cards. In *Smart Card Programming and Security*, volume 2140 of *LNCS*. 2001.
20. O. Reparaz, B. Bilgin, S. Nikova, B. Gierlichs, and I. Verbauwhede. Consolidating Masking Schemes. In *CRYPTO 2015*, 2015.
21. M. Rivain and E. Prouff. Provably Secure Higher-Order Masking of AES. In *CHES 2010*, volume 6225 of *LNCS*. 2010.
22. E. Trichina. Combinational logic design for AES subbyte transformation on masked data. *IACR Cryptology ePrint Archive*, 2003, 2003.